[IMDM327] Boids

Finding inspiration for this project was difficult at first, until the floodgates of Boid and simulation content were opened to me from just a few YouTube searches. Not long after watching a few simple code-based flocking videos, my browser recommended one I had seen many years ago. I remember this particular video because of its (at-the-time) incomprehensible stunningness. The creator Sebastion Lague had made a handful of small lines that, when combined, could form beautiful, organic, and mesmerizing patterns. When I first clicked to rewatch, I hadn't realized that the agents that created the slime simulation were quite similar in behaviour to the boids we had learned in class. In fact, the video began with an ant simulation with a boid-like architecture. While the logic that comprised both models shown in the video made sense on paper, I couldn't comprehend how Lague was able to have so many agents. The next day, I went back to it to find out what language it was in. I was shocked to find it was in Unity! Yet, it wasn't in C#, at least not entirely. Lague used a process I was unfamiliar with: compute shaders. Almost immediately, I knew I had found my motivation to make as many boids as I could. I wanted to achieve similar levels of scale and abstraction through number. I knew I'd color it like water, with white rapid particles, and dark blue slower ones, and a finger-like cursor to push through it.

The process of learning how to create my own shader compute shader required an understanding of a few key points. A useful analogy I found was that if the CPU is an airplane, which can quickly bring a small amount of luggage anywhere, the GPU is like a cargo ship, slower and only able to travel in water, but with significantly more luggage at once. In practice, this meant iterating over an array wasn't that, but dividing it into packaged chunks to be run simultaneously. On a slightly lower level, the C# script can dispatch multiple thread groups composed of any number of identifiable threads, each running the same kernel (function) at nearly the same time. I mapped each thread ID to an index in my list of boids that was read/write on both the C# and HLSL sides as a buffer. Other variables set in C# were passed as readonly to the shader. The boid logic is fairly standard, operating on 5 main parameters: neighbor radius (range of effect), separation distance, and scalars for cohesion, separation, and alignment. Cohesion and alignment served to group boids together by interpolating each boid's direction and position to the group's average by a factor of their respective scalars. Separation applied a change to the velocity away from other boids if under the separation distance. Fine-tuning these values with low separation distance and high alignment weight allowed me to reach a water adjacent consistency that I'd imagined. However, I wasn't satisfied with hard edges to the screen, nor just modding the position when out of bounds. Something I had not seen in Lague's video was looping boundaries. I later learned this creates a simulation space in the shape of a torus. In the future, I would like to try graphing it in 3D by mapping its linear x, y space to theta, theta space.

Sebastian Lague Slime Simulation: https://www.youtube.com/watch?v=X-iSQQgOd1A